

THE UNIVERSITY OF AUCKLAND

First Semester, 2002

City Campus

Computer Science

Language Implementation

(Time allowed TWO hours)

FAMILY NAME:
PERSONAL NAMES:
STUDENT ID NUMBER:
SIGNATURE:
LOGIN NAME:

This Examination is out of 70 Marks. Attempt **ALL** questions. Write your answers in the spaces provided in this booklet.

1		12
2		6
3		14
4		12
5		8
6		10
7		8
Total		70

Print Name _____

1. Bottom Up LALR(1) Parsing**[12 Marks]**

Consider the CUP grammar:

terminal LEFT, RIGHT, OR, STAR; /* “(”, “)”, “|”, “*” */

terminal Character CHAR; /* A single character other than “(”, “)”, “|”, “*” */

non terminal ExprNode OrExpr, ConcatExpr, SequenceExpr, SimpleExpr;

start with OrExpr;

OrExpr ::=

```

    OrExpr: expr1 OR ConcatExpr: expr2
    {
        RESULT = new OrNode( expr1, expr2 );
    }

```

|

```

    ConcatExpr: expr
    {
        RESULT = expr;
    }

```

;

ConcatExpr ::=

```

    SequenceExpr: expr1 ConcatExpr: expr2
    {
        RESULT = new ConcatNode( expr1, expr2 );
    }

```

|

```

    /* Empty */
    {
        RESULT = null;
    }

```

;

SequenceExpr ::=

```

    SimpleExpr: expr STAR
    {
        RESULT = new OptSeqNode( expr );
    }

```

|

```

    SimpleExpr: expr
    {
        RESULT = expr;
    }

```

;

SimpleExpr ::=

```

    CHAR: chr
    {
        RESULT = new CharNode( chr.charValue() );
    }

```

|

```

    LEFT OrExpr: expr RIGHT
    {
        RESULT = expr;
    }

```

;

Continued ...

Print Name _____

- (a) Using the information provided in the appendix, perform a shift-reduce LALR(1) parse of the input

$ab*c$

Note that the tokens are the individual characters “a”, “b”, “*”, “c”.

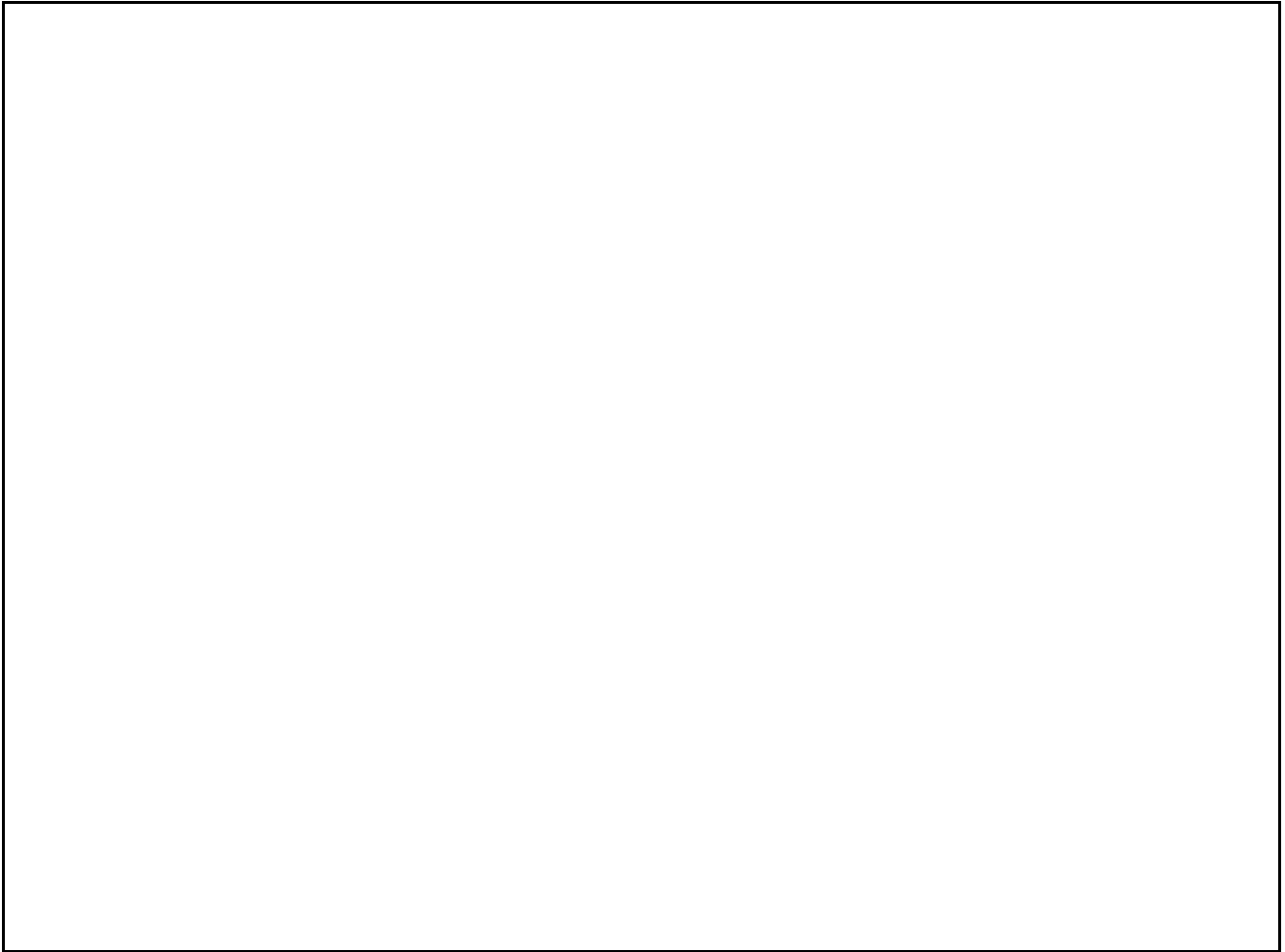
Stack				Token	Action
\$ 0				CHAR a	Shift 6
\$ 0	CHAR 6			CHAR b	SimpleExpr \square CHAR
\$ 0					
\$ 0					
\$ 0					
\$ 0					
\$ 0					
\$ 0					
\$ 0					
\$ 0					
\$ 0					
\$ 0					
\$ 0					
\$ 0					
\$ 0	Or 3				Shift \$ 8
\$ 0	Or 3	\$ 8			\$Start \square OrExpr \$
\$ 0	\$Start -1				Accept

(8 marks)

Continued ...

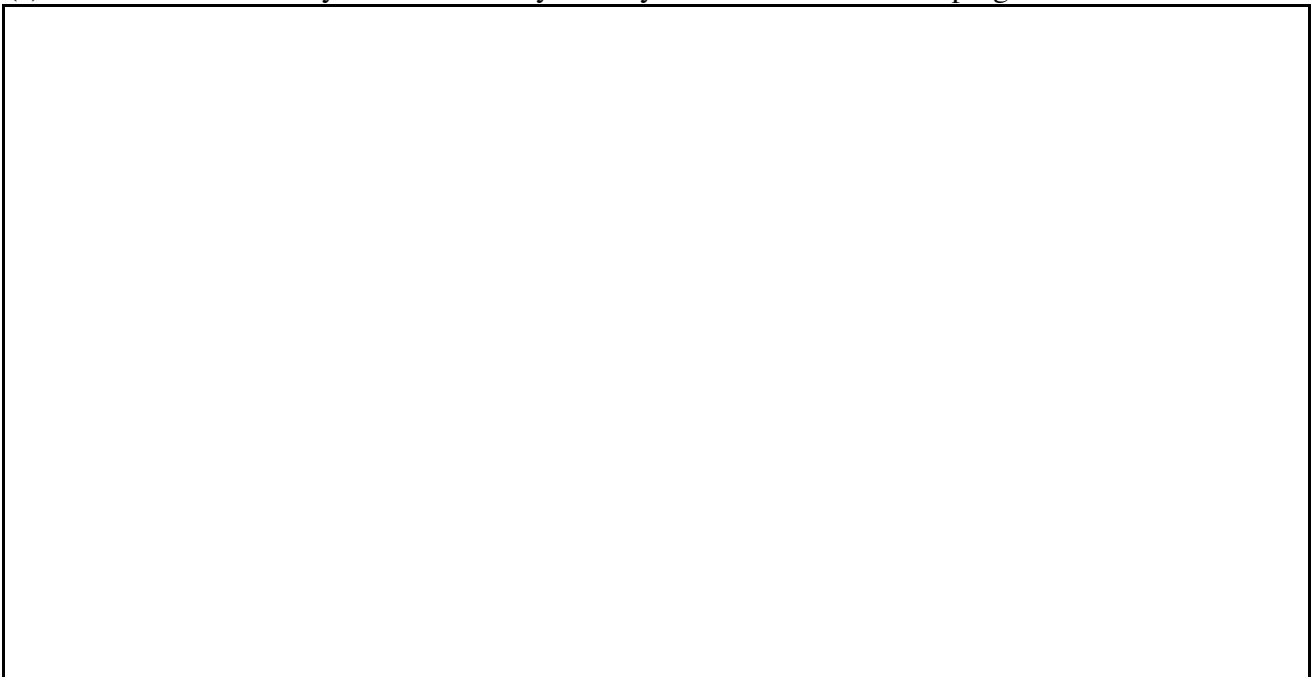
Print Name _____

(b) Draw the parse tree corresponding to the grammar rules used to parse this input



(2 marks)

(c) Draw the abstract syntax tree actually built by the actions in the CUP program



(2 marks)

Continued ...

Print Name _____

2. JLex**[6 Marks]**

Write a JLex program that

- Matches “{”s and “}”s, and records the number of “{...}”s it is currently nested inside. You may assume “{...}”s outside string constants are properly nested.
- Matches string constants, where a string constant is an (opening) double quote, 0 or more characters (excluding double quote and backslash characters), then a (closing) double quote. Thus escape sequences are not allowed inside string constants, but “{”s and “}”s might occur. For example, "{" is a legal string, but "", "\", and "\033" are not. “{”s and “}”s inside string constants should not change the nesting level. You may assume all string constants are well formed (i.e., obey these rules).
- Matches UNIX, Macintosh and Windows style line breaks.
- Matches any other character outside a string by a “default” rule.
- Reprints any text (string constants, line breaks, and other text) not enclosed inside “{...}”s.
- Does not reprint “{”s and “}”s or text enclosed inside “{...}”s.

Note that you do not need to use states to write this JLex program, although you do need an integer variable representing the current nesting level.

Continued ...

Print Name _____

3. Write a grammar definition**[14 Marks]**

Suppose we have a computer language, in which we can declare type identifiers. For example, we might declare some types as in

```
type
  integer = int,
  bitArray = []boolean,
  rational = struct( int numerator, denominator ),
  complex = struct( rational x, y ),
  complexArray = []complex,
  list = struct( int value; list next ),
  binaryTree = struct( int value; binaryTree left, right ),
  multiTree = struct( int value; []multiTree child ),
  day = enum( sun, mon, tue, wed, thu, fri, sat ),
  daySet = set( day ),
  charSet = set( char )
```

In this example, “integer” is declared to be the same as the basic type “int”. “bitArray ” is declared to be an array type with elements of the basic type “boolean”. “rational” is declared to be a structure (record) type with two integer fields called “numerator” and “denominator”, both of type “int”. “binaryTree” is declared to be a structure (record) with one integer field called “value” of type “int”, and two fields called “left” and “right” of type “binaryTree”. “day” is an enumerated type with members “sun” ... “sat”. “daySet” is a set type with element type “day”, etc.

Write a (general) CUP grammar definition for type declarations, capable of matching the complete input in the above example.

Assume:

- “type” is a reserved word, not an identifier.
- Existing types, such as int, boolean, char, are considered to be identifiers.
- An array type is represented by putting “[]” in front of the element type.
- A structure type is represented by enclosing a list of field declarations separated by semicolons, inside “struct(...)”.
- A field declaration is composed of a type, then a comma separated list of identifiers.
- An enumerated type is represented by enclosing a comma separated list of identifiers inside “enum(...)”.
- A set type is represented by a type enclosed inside “set(...)”.
- Semicolons (“;”) separate declarations (as in “int value; binaryTree left, right”), and do not terminate declarations.

You only have to write the grammar. You do not have to write any actions.

Continued ...

Print Name _____

Print Name _____

Print Name _____

4. Show the run time stack.**[12 Marks]**

Below is a program in the assignment 4 OBJECT3 language. Complete the drawing of the data structure built for the global variable “node”, and the run-time stack when the maximum level of nesting of method invocations occurs within the statement “insert(0, node, 18);” on line 44, and the process is almost ready to return. Remember the stack grows “up” towards low memory. Enter the appropriate values for each stack frame (activation record) you draw. Assume that all variables are implicitly initialised to zero or null. The line numbers on the left-hand side of the program should be used to represent the value of the program counter. Draw appropriate arrows for the var parameters, and pointers to class objects. Make sure you indicate the exact field pointed to in an object very clearly. The stack information has already been filled in for the main program, and most of the first stack frame. You may label addresses, and refer to them, rather than drawing arrows, if you need to, to make your answer clearer. Represent nodes as shown in the sample entry.

Also indicate the output generated by the complete execution of the program.

```

1   class BinTree
2       begin
3           instance
4               int value;
5               BinTree left, right;
6       end
7
8   void createLeaf( int level; var BinTree node; int value; )
9       begin
10          node = new BinTree;
11          node.value = value;
12      end
13
14  void insert( int level; var BinTree node; int value; )
15      begin
16          if node == null then
17              createLeaf( level + 1, node, value );
18
19          elif value < node.value then
20              insert( level + 1, node.left, value );
21
22          elif node.value < value then
23              insert( level + 1, node.right, value );
24
25          end
26      end
27
28  void printTree( BinTree node; )
29      begin
30          int i;
31          if node <> null then
32              printTree( node.left );
33              print( node.value + " " );
34              printTree( node.right );
35          end
36      end
37
38  BinTree node;
39
40  insert( 0, node, 52 );
41  insert( 0, node, 14 );
42  insert( 0, node, 23 );
43  insert( 0, node, 44 );
44  insert( 0, node, 18 ); // Display at maximum level of recursion
45
46  printTree( node );

```

Continued ...

Print Name _____

Top Of Stack (Low Memory)

Method Name:

Return Address:

Actual Parameters:

Method Name:

Return Address:

Actual Parameters:

Method Name:

Return Address:

Actual Parameters:

Method Name:

Return Address:

Actual Parameters:

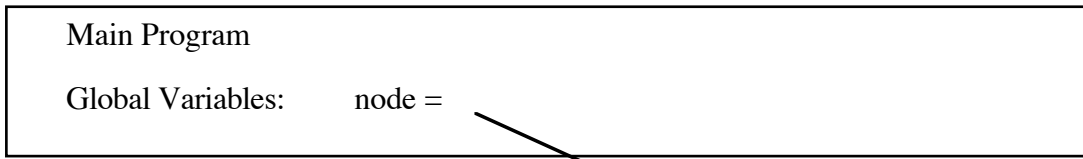
Method Name: insert

Return Address: Line 45

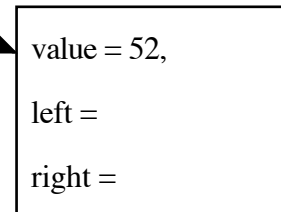
Actual Parameters: level = 0, node = , value = 18

Continued ...

Print Name _____



Bottom Of Stack (High Memory)



Output generated:

Continued ...

Print Name _____

5. Type checking and Evaluation**[8 Marks]**

Suppose we use an expression of the form “IDENT.IDENT” (for example “day.tue”) to select a member constant of an enumerated type.

Complete the code for the class representing such an expression, to implement an interpreter.

```
public class MemberNode extends ExprNode {  
    private String typeIdent, memberIdent;  
    private Env env;  
    private Decl typeDecl, memberDecl;  
    private EnumType enumType;  
  
    public MemberNode( String typeIdent, String memberIdent ) {  
        this.typeIdent = typeIdent; this.memberIdent = memberIdent;  
        precedence = 8;  
    }  
  
    public String toString() {  
  
    }  
  
    public void genEnv( Env env ) {  
  
    }  
  
    public Type checkType() {  
  
    }  
}
```

Continued ...

Print Name _____

}

```
public RunValue eval( RunEnv runEnv ) {
```

```
}
```

```
}
```

Print Name _____

6. Implementation of object oriented languages**[10 Marks]**

Suppose we have the Java program

```
class A {
    int c, d;
    A( int c, int d ) { this.c = c; this.d = d; }
    String e() { return "A.e()"; }
    String f() { return "A.f()"; }
    String f( int x ) { return "A.f( " + x + " )"; }
    static String h() { return "A.h"; }
}

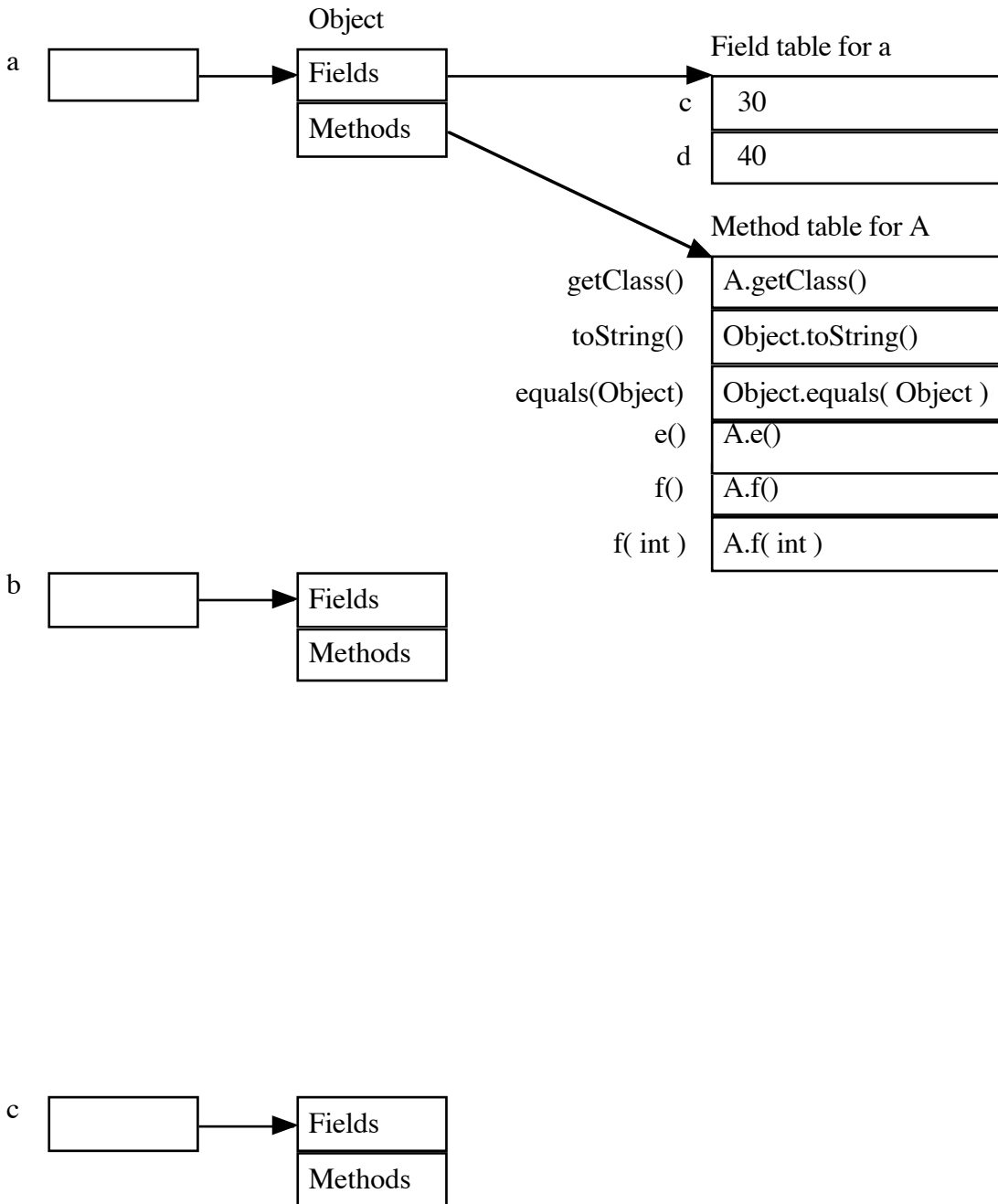
class B extends A {
    int c, e;
    B( int q, int r, int c, int e ) {
        super( q, r );
        this.c = c; this.e = e;
    }
    String d() { return "B.d()"; }
    String f() { return "B.f()"; }
    String f( char x ) { return "B.f( \' " + x + " \' )"; }
    String f( int x ) { return "B.f( " + x + " )"; }
    static String h() { return "B.h"; }
}

class Main {
    public static void main( String[] args ) {
        A a = new A( 30, 40 );
        B b = new B( 150, 160, 170, 180 );
        A c = new B( 210, 220, 230, 240 );
        System.out.println( "a.c = " + a.c );
        System.out.println( "a.d = " + a.d );
        System.out.println( "b.c = " + b.c );
        System.out.println( "b.d = " + b.d );
        System.out.println( "b.e = " + b.e );
        System.out.println( "c.c = " + c.c );
        System.out.println( "c.d = " + c.d );
        System.out.println( "c.e() = " + c.e() );
        System.out.println( "c.f() = " + c.f() );
        System.out.println( "b.f( 'A' ) = " + b.f( 'A' ) ); // 'A' is ASCII 65
        System.out.println( "c.f( 'A' ) = " + c.f( 'A' ) ); // 'A' is ASCII 65
        System.out.println( "c.f( 65 ) = " + c.f( 65 ) );
        System.out.println( "c.h() = " + c.h() );
    }
}
```

Continued ...

Print Name _____

- (a) Draw a diagram showing the data structures (object, field table and method table) created for the variables b and c, within the method Main.main. Shared data structures should be drawn only once.



Print Name _____

(b) Indicate the output generated by the method Main.main.

a.c = 30

a.d = 40

b.c =

b.d =

b.e =

c.c =

c.d =

c.e() =

c.f() =

b.f('A') =

c.f('A') =

c.f(65) =

c.h() =

Print Name _____

7. Code generation**[8 Marks]**

Assume objects are stored in a manner compatible with the diagrams displayed in question 6. Assume the current object is pointed to by the “\$ip” register, the current stack frame (activation record) is pointed to by “\$fp”, and the top of stack is pointed to “\$sp”. Assume that the actual parameters are passed on the stack.

Suppose “f” is an instance method of “a”, and “a” is an instance field of the current object (“this”). Indicate the assembly language likely to be generated to invoke “a.f(5, 6, 7)”. Use symbolic names “a” and “f” for the offsets of the corresponding instance field and method from the base of the relevant field and method table. Add comments to explain the purpose of your code.

_____ End of Questions _____

Continued ...

Print Name _____

This Page is left blank for questions that overflow

Appendix

Grammar Rules

8: SimpleExpr ::= LEFT OrExpr RIGHT
 7: SimpleExpr ::= CHAR
 6: SequenceExpr ::= SimpleExpr
 5: SequenceExpr ::= SimpleExpr STAR
 4: ConcatExpr ::=
 3: ConcatExpr ::= SequenceExpr ConcatExpr
 2: OrExpr ::= ConcatExpr
 1: OrExpr ::= OrExpr OR ConcatExpr
 0: \$START ::= OrExpr EOF

Action Table

From state #0
 EOF:REDUCE(4) LEFT:SHIFT(1) OR:REDUCE(4)
 CHAR:SHIFT(6)
 From state #1
 LEFT:SHIFT(1) RIGHT:REDUCE(4) OR:REDUCE(4)
 CHAR:SHIFT(6)
 From state #2
 EOF:REDUCE(4) LEFT:SHIFT(1) RIGHT:REDUCE(4)
 OR:REDUCE(4) CHAR:SHIFT(6)
 From state #3
 EOF:SHIFT(8) OR:SHIFT(9)
 From state #4
 EOF:REDUCE(6) LEFT:REDUCE(6) RIGHT:REDUCE(6)
 OR:REDUCE(6) STAR:SHIFT(7) CHAR:REDUCE(6)
 From state #5
 EOF:REDUCE(2) RIGHT:REDUCE(2) OR:REDUCE(2)
 From state #6
 EOF:REDUCE(7) LEFT:REDUCE(7) RIGHT:REDUCE(7)
 OR:REDUCE(7) STAR:REDUCE(7) CHAR:REDUCE(7)
 From state #7
 EOF:REDUCE(5) LEFT:REDUCE(5) RIGHT:REDUCE(5)
 OR:REDUCE(5) CHAR:REDUCE(5)
 From state #8
 EOF:REDUCE(0)
 From state #9
 EOF:REDUCE(4) LEFT:SHIFT(1) RIGHT:REDUCE(4)
 OR:REDUCE(4) CHAR:SHIFT(6)
 From state #10
 EOF:REDUCE(1) RIGHT:REDUCE(1) OR:REDUCE(1)
 From state #11
 EOF:REDUCE(3) RIGHT:REDUCE(3) OR:REDUCE(3)
 From state #12
 RIGHT:SHIFT(13) OR:SHIFT(9)
 From state #13
 EOF:REDUCE(8) LEFT:REDUCE(8) RIGHT:REDUCE(8)
 OR:REDUCE(8) STAR:REDUCE(8) CHAR:REDUCE(8)

Reduce (Go To) Table

From state #0:
OrExpr:GOTO(3)
ConcatExpr:GOTO(5)
SequenceExpr:GOTO(2)
SimpleExpr:GOTO(4)

From state #1:
OrExpr:GOTO(12)
ConcatExpr:GOTO(5)
SequenceExpr:GOTO(2)
SimpleExpr:GOTO(4)

From state #2:
ConcatExpr:GOTO(11)
SequenceExpr:GOTO(2)
SimpleExpr:GOTO(4)

From state #3:
From state #4:
From state #5:
From state #6:
From state #7:
From state #8:
From state #9:
ConcatExpr:GOTO(10)
SequenceExpr:GOTO(2)
SimpleExpr:GOTO(4)

From state #10:
From state #11:
From state #12:
From state #13:

End of Appendix